CHRISTOPHER D. LEIDIGH

# Configuring TCP/IP Hosts, Methods, and Protocols

Every TCP/IP-enabled device needs to be configured. Of course, there are several valid ways to go about it. This article will help you select the best configuration technique for your system.

Everything needs to be configured by someone. The fact of the matter is that configuration of computers and some embedded devices continues to be a hassle. Embedded systems can actually be some of the worst culprits in this regard, due to the fact that many tend to be "interface-challenged." (Need I mention the blinking "12:00" on your VCR?) Any system, embedded or otherwise, participating in a TCP/IP network requires at least some minimal configuration information. I'll refer to this as "IP configuration," given that the parameters are IP configuration elements. Currently, six basic configuration methods or schemes are in use. While I'm sure that others may exist, those presented here are used in the mainstream. The six methods are static local assignment, preset static unicast or multicast addresses, *Reverse ARP* (RARP), *gleaning*, *Bootstrap Protocol* (BOOTP), and *Dynamic Host Configuration Protocol* (DHCP). These will be discussed along with their uses and limitations.

## Basic IP configuration information

Any host "actively" participating in a TCP/IP network requires at least three basic IP configuration elements: a network-unique IP address, a subnet mask, and a default gateway IP address. By "actively" we mean to say that our host needs to be able to communicate with other nodes on the network using the TCP/IP protocols.

If a host were just a listening device, such as a network monitoring tool or probe, then no such information would be required. Such a device usually listens on a network in a passive, "promiscuous" mode (see Thomas Herbert's

"Introduction to TCP/IP," December 1999, p. 57) and records and displays all network traffic. For our purposes, we assume that we actually want to communicate. Therefore, we need to configure each network-connected device such that it has a unique IP identity.

Of the three elements, the *IP address* is the most important. This is the 32-bit address of the device and it must be unique on the network in which the host resides. If the host participates in a publicly routed network (that is, attached to the Internet), then the address must be globally unique. In general, this means that the address is within the range of addresses for an Internet-registered network. IP addresses are usually written in *dotted decimal notation* such as 159.215.6.40, with the dots separating bytes. IP addresses are actually composed of three parts: a *network id* portion, a *subnet id* portion, and a *host id* portion.

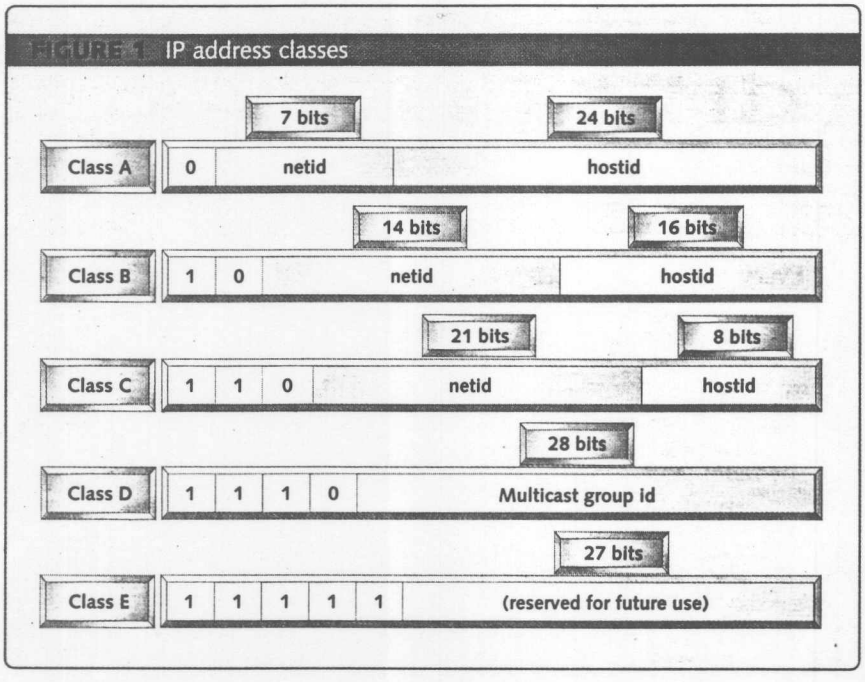
IP addresses are registered and assigned by the Internet Addressing Board (IAB) and are categorized into *network classes*. The different classes are listed in Figure 1. The class table shows that the different class networks define the maximum number of bits that can be used to describe the subnet and host portions of an address within a network. For example, the IP address 159.215.6.40 is on a Class B network with a network id of 159.215, the 6.40 represents the subnet/host id portion. (This decoding is accomplished with the aid of the *subnet mask*.)

The subnet mask is a separate IP configuration element that is a bit mask for the network id and subnet id portion of an address. While not a strict requirement, subnet masks are typically chosen on byte boundaries so that it is easy to read IP addresses and determine what subnet they are on. If 8-bit subnets are used for the example network address, then the subnet mask would be 255.255.255.0. Since the Class B address uses the top 16-bits (159.215), the subnet can be obtained by a bitwise OR of the IP address and subnet mask, and subsequently bitwise ANDing out the top 16 bits of the network id for a result of 0.0.6.0, the "six" subnet.

Subnets are used in TCP/IP to logically divide networks into smaller portions. Dividing networks is necessary to segment traffic and provide for a means of associating locations with subnets. It is the job of *routers* or *gateways* to move TCP/IP packets between IP subnets on an as-needed basis. For example, to send an e-mail from one company to another, a message would have to pass from Company A to the Internet and into Company B. Such a journey would typically involve many "router hops" as the message moved between Company A's subnets, portions of the Internet, then Company B's subnets.

The *default gateway* configuration parameter is used by TCP/IP hosts that need to communicate with other systems that are not on the same IP subnet. Packets destined for an off-subnet address will be forwarded by the sending host to the default gateway. The gateway (router) will then forward the packet in the appropriate manner based on information it has from routing protocols. A TCP/IP host need not know the path to other hosts outside the local subnet, it can leave this job to the default gateway. An isolated LAN situation would actually not need either the subnet mask or default gateway since everything would be on the same IP subnet. Such a situation might occur in a small, isolated office LAN.

These three pieces of information serve as the "minimal" configuration for a typical TCP/IP host. However, by no means do they represent even a fraction of the total possible configuration elements. Almost all other elements would be associated with specific features or TCP/IP applications implemented on a host. This extended configuration information can either have well-known defaults or be configured by a user remotely over the network. The main difference is that the IP address, subnet mask, and gateway are all specific and unique to a user's network and the actual host. Therefore, they cannot have defaults.




## Static local assignment

There is no official networking term "static local assignment." Some texts do use the term "static assignment" for

this configuration method. However, "local" is added here to distinguish this method from the remote static methods described later. In general, most hosts will have a method for static local assignment of IP configuration information. However, it is assumed that there must be some form of a user interface to do so. While PCs have screens and keyboards, embedded systems are typically limited to serial ports, LCD screens and buttons, or even binary switches. For embedded hosts the most typical static configuration scenario would require a user to start a terminal program connected to a serial port and configure the device's parameters through a menu or command line interface.

The static local assignment method is by far the most widely used when considering all TCP/IP hosts. While certain categories of devices may utilize other methods to a greater extent, static local assignment is administratively the easiest to use and implement in most situations. Hosts can be centrally configured and then deployed to their final destinations where they can expect to boot up and be usable. Of course, other configuration information could be assigned in a local manner as well.

The key question is really when not to use static local assignment of IP configuration information. The answer to this question depends on two general issues: how and where the device is used and how it is deployed. For example, at the company I work for, we offer network management cards for many of our uninterruptible power supply (UPS) products. At a large customer's site, hundreds of these may be deployed at once. Consider two different deployment methods:

1. The customer has a centralized configuration strategy where equipment is staged and fully configured before it is deployed to its final location
2. The customer drop-ships "raw" unconfigured equipment to the deployment sites, where it is subsequently installed by field teams

The first scenario would require each device be configured one by one through the serial port, a time-consuming task when hundreds of units are involved. This method, however, requires only one central configuration team versus many experienced field teams. The second scenario could actually utilize either local static assignment by the field teams or one of the dynamic or remote methods explained later.

## Static, preset unicast, or multicast addresses

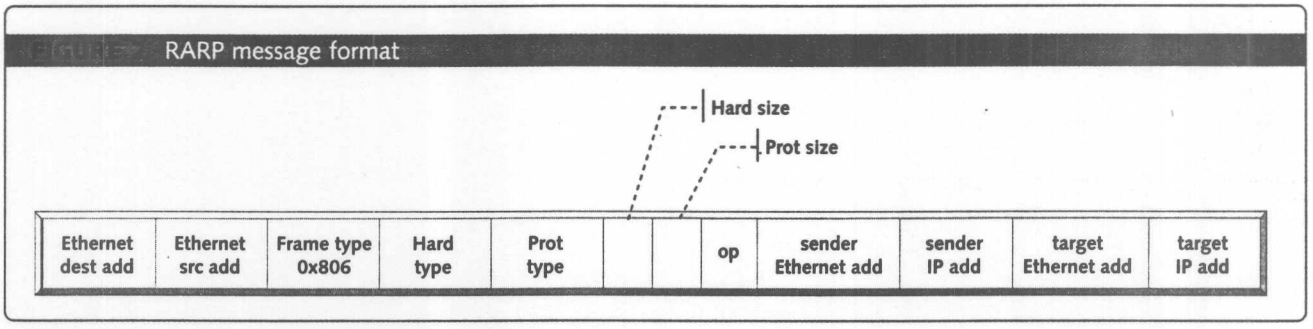
Using factory preset addresses can eliminate the need for direct, local address assignment described above. These methods can accomplish over-the-network configuration in some circumstances. Assuming that a network host to be installed has a factory default address of 192.78.78.66, a user can temporarily change the address of their computer to be on the same pseudo subnet for the purpose of configuration, perhaps using address 192.78.78.67. After doing so, the user can use built-in access protocols, such as telnet, to access the new host and configure it appropriately for the local network. Following this, the user would change the configuration computer's settings back to normal so that both the new device and the computer could communicate on the operational network.

A small variant to this method utilizes an IP multicast address as the factory pre-configured address. Using a multicast address has the advantage of not requiring an address change to the local computer used for configuration. The reason behind this is that IP multicast addresses are independent of IP subnets and, therefore, the configuration computer can communicate with the host's preset multicast address with no address change.

## Reverse ARP (RARP)

In the "old" days (when I was still in college), many terminals that connected to TCP/IP networks were both diskless and without local interfaces. (Remember when X was the only kind of windows?) Not only did these systems lack local nonvolatile storage for applications, but they typically did not store their IP configuration parameters either. In these systems it was necessary to obtain IP configuration information "dynamically," at bootup. Reverse ARP (RARP) based on the *address resolution protocol* (ARP), was designed to accommodate such systems. The protocol is described in RFC 903. The ARP protocol is used to bind hardware addresses—also referred to as Media Access Control (MAC) addresses—to IP addresses. When a host needs to communicate with another TCP/IP host, it broadcasts an ARP message requesting the owner of the destination IP address to respond with its hardware address. The RARP protocol is just the reverse lookup; given a host's hardware address a request can be made for the corresponding IP address.

A RARP server is a process that can run on one or more host server systems in a local IP subnet. These servers contain configuration files that specify IP addresses for RARP clients. Each entry would contain an IP and hardware address pair. Host systems are configured to use RARP to obtain an IP address by broadcasting an RARP request at startup. The RARP message, shown in Figure 2, contains the client's hardware address and network type (Ethernet, Token Ring, or FDDI). Given that the request is sent as a link layer broadcast, all nodes will receive the request and the client doesn't need to know the addresses of RARP servers explicitly. Upon receipt, a RARP server will look up the client's hardware address in its tables. If a match is found, the server will send a directed response message, a RARP reply, specifying the IP address for the client. Multiple RARP servers can be used for redundancy or other reasons. If several RARP servers have an entry for the same client they will both reply to a request and the client will just process the first reply. For the sake of reducing network traffic, servers can be configured as primary and secondary. Secondary servers would only respond if the primary were down and a second client request was received.




While this method eliminates cables and local interfaces, it is not always practical or desirable to have to change a computer's address information just to configure a new network device. For a very small deployment this may be reasonable, but it is not a possibility for a large rollout where installers may not be well trained. The requirement of being on the same IP subnet also precludes this method from being used remotely. Another limitation is that only one device can be added to the network and configured at a time to avoid simultaneous duplicate addresses (all the units would have the same default factory address).

While RARP is simple to implement and use, this configuration mechanism is only capable of supplying the IP address. The subnet mask and gateway would have to be configured in another manner. Why would such a mechanism be used given the limitations? The advantages of RARP lie in the fact that

nodes need not be directly configured for deployment. Connecting cables and accessing configuration menus are avoided. In the case of embedded systems where interfaces such as keyboards and screens usually do not exist, this can be much easier for installation. Network managers can record the hardware addresses of hosts, place them in a RARP configuration file along with IP addresses and then boot up the hosts. Subsequent IP address changes can also be made by changing the RARP configuration files and rebooting the RARP client. When the client restarts, it makes another RARP request and receives the new address.

Given the fact that only an IP address can be obtained through RARP, additional information must be configured through other means such as a telnet session to the host. Telnet is a basic remote console program that usually provides access to a host's command line interface over the network. Through the use of this command line interface, a user could configure the remaining items.

RARP can be categorized as one of several dynamic, over-the-network configuration methods. As one of the tenets of a dynamic configuration scheme, the host should not have to be configured or accessed prior to deployment. In the case of RARP, only the hardware address need be noted for placement in the RARP configuration file. The broadcast of the request message avoids the need to know about specific RARP servers, which would require preconfiguration. An important limitation, however, is that the broadcast message will only be sent on the local subnet as routers do not forward broadcasts. This limits the use of RARP to the local subnet. That is both the clients and servers must be on the same subnet.

## Gleaning

The method of IP address assignment through gleaning is not as widely known or implemented as the other schemes described here. One of the few devices to use such a scheme is the Axis line of print servers. Gleaning is also a dynamic method requiring no prior configuration of a device. It also takes advantage of the fact that hosts have unique hardware addresses.

There is no RFC describing the gleaning mechanism and, therefore, there is not necessarily one way to implement gleaning. The basic premise is that if a host is in an unconfigured state—that is with no IP address information—and it receives a link layer directed packet, the host can "glean" its IP address from the IP address in the packet. A step-by-step explanation is as follows:

1. The user places the hardware address of the host being setup into his/her workstation or PC's ARP cache through a command such as *arp –s 00:c0:b7:44:88:99 159.215.6.22*. This command, which is available in various forms on almost any Unix and Windows system, gives the PC the hardware address to IP mapping usually obtained through an ARP request message over the network. The IP address argument should be that which is intended for the host. Placing the entry in the cache manually *prevents* the need for the ARP message because the PC already has the mapping. Since the host being setup by this method does not have an IP address, it could not and would not respond to an ARP request.
2. Subsequent to "priming" the ARP cache, the user issues a *ping 159.215.6.22* command to send an ICMP echo request to the device. Normally, the PC would have to first issue an ARP request to find out the hardware address of 159.215.6.22 before sending the ICMP echo request message. Given that the ARP cache has the entry that was manually entered, it can issue the echo request immediately. The UDP/ICMP message that gets sent has a destination hardware address of *00:c0:b7:44:88:99* and an IP address destination of *159.215.6.22*. Since the link layer driver on a system will receive all packets explicitly addressed to the device's hardware address, the packet will be brought into the system. The link layer driver would then pass the packet to the first layer of the TCP/IP stack, the *network* or *IP* layer, which would then determine if the packet's destination IP address matches that of the device. At this point the special gleaning code detects that the device is unconfigured *and* has received a MAC and IP directed (unicast) for the ICMP echo request. These three factors allow the device to "glean" that it is being requested to set its IP address to that of the destination IP address of the ICMP echo request packet.
3. Once the device sets its IP address accordingly, it also goes out of the *unconfigured* state. This prevents gleaning from occurring again. At this point the device is partially configured. We still do not have the other two minimal elements, the subnet mask and the default gateway. So what have we accomplished? In this partially configured state we can begin communicating with the device as long as we are on a PC (or other host) on the same IP subnet. Following this restriction we actually alleviate the need for the subnet mask or default gateway because we are not involving off subnet communication (which requires the default gateway address). The effect of the lacking subnet mask is subtler. Usually the subnet mask is used to determine whether a target IP address is on or off subnet as described earlier. When the device has no subnet mask it just assumes all communication is on the same subnet and sends packets directly to the target addresses and not to a gateway.
4. A user can now *telnet* to or web browse to the host, depending

upon the protocols supported. Once connected, both the subnet mask and default gateway can be setup through a menu or command line method.

Gleaning is a clever way of doing a minimal configuration (IP address only) without the need for cables, prior setup, or any user installed software. The scheme only requires an ARP process that can accept manual entries and the ubiquitous PING program. In fact, other variants can be implemented that utilize telnet perhaps instead of PING. The designer of the device can choose a variety of accepted directed packets that signal the configuration.

It is important to note that in all circumstances either the link layer or network layer must have the gleaning code added to it that does the detection and configuration. This assumes that the designer of the device has access at that level directly, perhaps modifying existing stack source, or indirectly through appropriate callbacks. The latter case is not likely to exist in any off-the-shelf TCP/IP stack. Those who modify their TCP/IP stack should do so only if they fully understand how the stack works.

## BOOTP

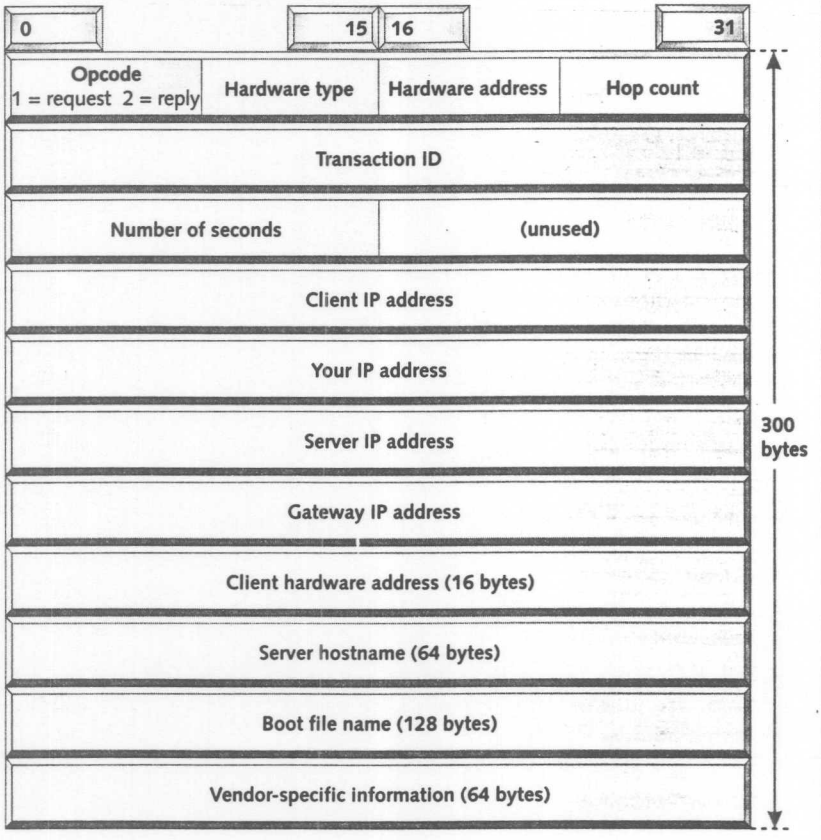
The limitations of RARP as a configuration method led TCP/IP developers to design a more flexible configuration protocol called BOOTP (Bootstrap Protocol). Again, the original intent was to enable a dynamic, over-the-network configuration method for diskless host devices. The BOOTP protocol overcomes the RARP limitation of only configuring IP addresses and is not limited by link layer broadcasts.

RFC 968 describes the BOOTP protocol in detail. The protocol is a simple client-server, request-response mechanism similar to RARP. BOOTP, however, utilizes UDP datagrams over IP. In its simplest form, a BOOTP client makes a BOOTP request with certain client information and a BOOTP server responds with the necessary configuration information for the client. The request message is sent as an IP broadcast datagram since the BOOTP server addresses are not known. The "well-known" BOOTP UDP port numbers are used by both client and server, 67 in the case of the server and 68 for the client. BOOTP servers contain a configuration file, typically called the "boottab" for *boot table*, that contains the list of preconfigured BOOTP clients. Each entry has a hardware address and one or more configuration elements such as

FIGURE 3 Boottab file entry

```
.....................
Christopher_device:\
ha = 00:c0:b7:88:99:78:\
ip = 159.215.6.44:\
sm = 255.255.255.0:\
gw = 159.215.6.1
.....................
```

FIGURE 4 BOOTP message format




the IP address, subnet mask, and default gateway. In addition, it's possible to have other elements such as a host name, a boot file name and location, and so on. A typical boottab file entry is shown in Figure 3.

The BOOTP message format is given in Figure 4. The meanings of the fields are as follows:

- *Opcode* specifies request or reply (1 or 2, respectively)
- *Hardware type* indicates the requester's link hardware type (e.g., 10 for Ethernet)
- *Hardware address length* is the length of the hardware address (six for Ethernet)
- *Hop count* is used in the reply to tell how many routers (in a relay situation) the response has gone through
- *Transaction ID* is set by the client in the request and used in the server's reply to match request replies
- *Number of seconds* is the host uptime
- *Client IP address* is set by the client if it already knows its address (it may just seek a gateway)
- *Your IP address* is set by the server in the reply, to give clients IP addresses when they do not already have them
- *Server IP address* is the address of the server, set by the server
- *Gateway IP address* is the local router's IP address
- *Client hardware address* is set by client so that the server has easy access to it
- *Server hostname* is set by the server
- *Boot filename* is an optional boot file that the client can download from the server
- *Vendor-specific information* contains additional configuration options (up to 64 bytes)

Besides the standard information returned by BOOTP servers, the vendor-specific information section can contain additional configuration parameters. The first entry in this area, the "magic cookie" field, indicates whether vendor-specific options are present in the BOOTP response. When the magic cookie equals **99.130.83.99**, options are present. The vendor options are a series of entries including a one-byte *tag* followed by a length byte *n* and the actual *n* bytes of data. While the tags can be interpreted by the host, quite a few vendor-specific options have become fairly standard and are enumerated in RFC 1533. Two examples are shown in Figure 5.

BOOTP utilizes broadcast messages. This generally limits BOOTP clients and servers to be on the same IP subnet. Unlike RARP, though, BOOTP uses IP broadcasts. While these broadcasts are not forwarded across routers by default, a mechanism called a BOOTP Relay Agent can be run on routers if necessary.

BOOTP works well in those situations that configuration information is stored centrally and can be managed at the BOOTP servers. When changes are made, the clients can be rebooted to obtain their new information. The main limitation is that the hardware address of the devices must be tracked and entered, along with configuration information, into the BOOTP servers. If devices are deployed to incorrect locations, the corresponding configuration information will not work. In essence, while BOOTP can be consid-

ered a dynamic, over-the-network configuration protocol, there is a static setup at the BOOTP server.

## DHCP

The Dynamic Host Configuration Protocol (DHCP) is a superset of the BOOTP protocol and is defined by RFC 1541. DHCP extends the functionality of BOOTP in several areas; the vendor options area allows up to 312 bytes of configuration information and IP addresses can be "leased" from a DHCP server. The lease functionality of DHCP is the most significant addition and was designed to make address assignment to computers, especially laptops, more dynamic and automatic.

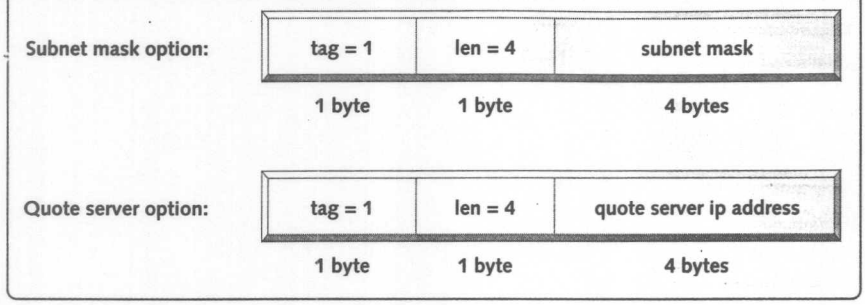
The DHCP request and response packets follow the same format as BOOTP with some DHCP-specific operation tags. We will limit the discussion here to the functional differences rather than going into detail on the message sequence and formats. The interested reader is encouraged to refer to the RFC for more detail.

The DHCP protocol allows for three different address assignment options. With *manual* assignment, a DHCP administrator can define an entry in a DHCP server configuration to respond to a specific client computer with a specific IP address configuration. This mode operates identically to BOOTP and, in fact, a DHCP server can serve BOOTP clients in this fashion.

The *automatic* address assignment mode differs in that the administrator need not preconfigure an address entry for a client. When an arbitrary client makes a request, an address is served to the client that it can use permanently from that point forward. In effect, this mode lies between a pre-configured, static assignment mode and the fully dynamic mode explained hereafter.

The *dynamic* address assignment mode introduces the concept of *leasing* IP addresses for some length of time to a DHCP client host. Specifically, a DHCP administrator configures a DHCP server to utilize one or more *IP address pools*. When a DHCP request is received, the configuration is checked to determine if the client is listed as either a static client or a previously known automatic client. If not, the server selects a "free" (currently unused) IP address from its pool. This address, along with other automatic configuration information such as subnet mask, gateways, and other utility servers, is returned to the client. Along with the address, however, is a *lease time*. The lease time tells the client how long it can use the address. Once the lease has expired, the client must make another DHCP request. Administrators can configure lease times from very short, per-

FIGURE 3 Vendor information field option formats




haps an hour, to infinity, which is a permanent assignment.

The DHCP protocol eases computer configuration because new computers (or devices) can be added to a network and be configured and start communicating automatically. No client information needs to be set up in a configuration file.

Perhaps the most significant use of DHCP is in facilitating configuration of mobile laptop users. Without DHCP, a traveling laptop user requires reconfiguation each time the laptop moves from one network location to another. A laptop configured to use DHCP (typical for the Windows operating system) makes a DHCP request each time it boots on a new network. The local DHCP server then supplies the laptop with the exact configuration information it needs for the local network including an IP address. The lease time allows the address to be freed and go back into the available pool when the laptop user leaves.

The lease mechanism allows a small address pool of dynamic addresses to be leveraged across many potential mobile users. The pool need only be as large as the number of simultaneous DHCP users. Without the automatic DHCP scheme, administrators are forced to look for a free address, give it to a user, and configure the laptop. If a user should leave and return without manually "re-requesting" an address from the administrator, there may be situations where address duplications on the network occur. Obviously, the human administration is both time-consuming and error-prone.

Although DHCP is extremely helpful to laptop users and network administrators, its uses for address assignment in embedded network devices are more limited. Consider the difference between a laptop and an embedded network device such as the network management card in a UPS. A laptop will typically have a unique host name given to it by a user or administrator, perhaps ChristopherLaptop1. Should this laptop be installed on a Windows network, it could receive a dynamic IP address through DHCP and broadcast its host name using TCP/IP-NETBIOS (a Windows networking protocol). Since it has an IP address, TCP/IP communication is facilitated and the laptop name will appear in the computer list in Windows Explorer. While the laptop user is connected and happy, other interested parties could actually locate and use the laptop by name as well.

The UPS device has some key differences. First, when the unit is initially deployed and installed, it will not have anything configured in it except factory defaults. Secondly, as an embedded device, we assume that the device is more "server-centric," that is it provides some service to the network or its users, the use of which is obtained by connecting to the device through some TCP/IP protocol such as telnet or HTTP (for browser-based access). The first issue limits our ability to "identify" the unit uniquely out of the box. The second point emphasizes the fact that as a user, we are not operating *from* the unit as in the laptop case, but rather to the unit which requires us to know the IP address of our device. If the device booted up, requested a DHCP address, and received one from the pool, how would we know what the address is? It would be possible for the device to advertise itself using NETBIOS and show up on a Windows list. However, this solution is Windows-centric and does not address the issue of unique identification.

Returning to the example, if we had multiple UPS systems on a local network, how would we distinguish between them? Without pre-configuration (the task we wish to avoid with DHCP), the UPS might have a model number, a serial number, and a hardware address. The latter two are unique, but not easy to figure out from a distance. At a minimum we would have to read the serial number from the back of the UPS and note it and the location of the UPS for later correlation. This scheme also assumes another piece of infrastructure is in place, a DHCP-to-DNS (Domain Name System) connection. Such a connection populates a DNS server with host-name-IP address pairs for dynamically configured devices. While such systems exist they are not standard or typical.

In the end, assuming we did everything mentioned, we could communicate with the UPS by using *telnet ups1400VA_SN2366590*. The host name "ups1400VA_SN2366590" would resolve to the IP address assigned by DHCP through an automatic DNS entry population. Inelegant at best and not much work saved over BOOTP at worst. This example highlights why DHCP is rarely, if ever, used to configure network infrastructure or embedded devices. Of course, those devices with a Windows relationship, such as network printers, are a noteworthy exception. Printers tend to be more easily identified and can be more directly managed through Windows network names.

## Different strokes

As should be evident, there is no one perfect or "best" configuration method. Different devices and different circumstances call for unique

installation and configuration choices. As a network device provider, it is most important to understand how your customers are going to deploy your devices and provide them with the requisite configuration options. In most cases, devices should have both a static local configuration method and one of the flexible, over-the-network configuration methods such as BOOTP or RARP. The latter two enable remote configuration and reconfiguration.

Pre-assigned addresses, gleaning, and DHCP tend to be most useful in circumstances where devices are being installed locally and are unique on the local network. In these cases, it is desirable to use the quick, cable-free methods. Device identification is not a problem because many identical units are not being installed.

An additional, similar, scheme is to provide a custom program that a user can install and use on a network computer that mimics a BOOTP or RARP server. Such a program could "listen" for BOOTP or RARP requests from your devices only. When a request is received, the user can be prompted for IP configuration information. The program could then respond, again as if it were a BOOTP or RARP server, and configure the device. Several companies employ such methods today.

While some of these protocols are indeed standards covered by RFCs, their exact use is not. Combining methods such as gleaning and BOOTP require the developer to determine the relationship between the configuration methods both in terms of sequencing as well as priority. For example, if a device receives an IP address through gleaning while it is making BOOTP requests, presumably the address should be used and the BOOTP process stopped. Another scenario could be that a device boots and makes BOOTP requests for a while, but receives no response—the device had received a BOOTP response at an earlier date. Should the device "fall back" to that prior address or remain unconfigured and inaccessible?

Such questions are but a few of the many that need to be answered when implementing a robust and flexible configuration scheme for your network embedded systems. **esp**

---

*Christopher Leidigh leads the embedded networking group at American Power Conversion, implementing SNMP and HTTP management platforms for power and environmental protection systems. He has been programming in various languages for over 14 years and has been doing embedded hardware and firmware design for eight years. He can be reached at CLeidigh@apcc.com.*

---